Active Advice Inverse Reinforcement Learning in Large State Spaces

Gonçalo Silva
Instituto Sistemas e Robótica (ISR)
Instituto Superior Técnico (IST)
Lisbon, Portugal
g.carrola.silva@tecnico.ulisboa.pt

***Abstract*—Autonomous agents usually need a decision system in order to know which data should be sent to the actuators depending on the data received in the sensors. This decision system works as the agent's brain, and can be obtained using several learning approaches. In learning from demonstration, the agent learns with the information given by a demonstrator, which shows correct behavior about how to proceed. The main problem of this approach is the need for too many demonstrations to learn large problems. This thesis aims to minimize this issue by using maximum likelihood inverse reinforcement learning followed by active query selection to learn large problems by demonstration and advice, minimizing the number of "trips" to the demonstrator. The implemented algorithm is applied to a grid-world and a grasping problem, which have a large discrete state and action sets. It is proposed a process of clustering of the states to label clusters in order to speed up the process of learning and a query selection method based on uncertainty to give to the demonstrator the most uncertain cluster to label.**

***Index Terms*—Inverse Reinforcement Learning, Reinforcement Learning, Learning from Demonstration, Active Learning, Advice Learning, Large State Spaces**

I. Introduction

Nowadays it is increasingly necessary to automate processes in order to facilitate people's lives and maximize the performance of tasks. The introduction of autonomous agents in carrying out tasks is an important step to perform that. These autonomous agents must be able to adapt to different situations that may occur outside of their routine. To do this, they need to analyze the environment where they are inserted, evaluate the possible consequences of each action they may take and decide for one that accomplishes in the best way what it is purposed. Basically, what it is wanted in this case is that an autonomous agent analyze a situation based on its knowledge and experience and then make a decision that maximizes the consequence intended.

A. Motivation

To achieve the purpose of taking the best decision to a certain situation, learning from a human that already has the optimal actions is a great and efficient solution. An agent benefits from learning with human demonstrations, instead of going through trial and error. The agent's decision system stores the knowledge taken by previous experience and with that information makes the decision he thinks it is the best for a certain situation. That is why learning from demonstrations seems so natural: a robot must learn from someone more experienced, in order to achieve optimal decisions faster.

B. Problem Statement

Learning from demonstration is a natural way of thinking about learning processes, as it is similar to the way humans learn. However, this approach has some particular problems that are not desirable in the area of robot learning.

First of all, a lot of training is necessary to obtain a strong model that allows the robot to know what it has to do in a given situation. In other words, the demonstrator will have to give to the robot a lot of demonstrations about the behavior of the environment. However, the agent can have limited memory resources, so it is crucial that this information be the most informative possible. The problem that is presented becomes worse for large state spaces, so it is necessary an even bigger amount of data to perform well. In large state spaces it is also difficult to perform enough experiments on all of the states. To fight this issue, the concept of Inverse Reinforcement Learning (IRL) was born, a concept that obtains a reward function from the currently known human demonstrations. To minimize the problems mentioned, it is crucial that the samples given to the robot are as informative as possible, so that less data is needed to obtain similar results. If the agent is not so certain of what action it should take in the respective state, then it is best to request to the user the correct information instead of unrelated demonstrations that end up becoming contradictory or redundant. This search for the most informative features is the goal of Active Learning (AcL) [1]. To choose the most informative features it is necessary to first evaluate them according to its informativeness.

Another problem consists of the possible errors made by the demonstrator. The demonstrator is usually an human being, and like all humans it can make mistakes. This issue must be considered, because it can lead to contradictions on some demonstrations and the learning iteration will not be as much effective as we expect. Advice Learning (AdL) makes it possible to provide information to the agent that is less susceptible to errors, thus being able to correct possible suboptimal demonstrations. Advice in this context is given by a set of preferences in state/action/reward spaces. Advice differs from demonstration in the sense that advice can cover more information about the state, namely whether a certain

feature is beneficial or detrimental to the problem, whereas a demonstration only refers to the best action for a certain state.

C. Objectives

The goal of this work is to speed up the process of learning from demonstration, applying methods that make the policy converge to optimal more quickly in large state spaces.

To perform that, it will be used some techniques to maximize the informativeness of the "lessons" given by the user, by focusing on the areas of the state space that the agent does not know so well. If we focus on those less certain areas to request and then learn, we will need less demonstrations to perform a good policy. The algorithm will also incorporate information in the advice format to complement learning by demonstration and we will see the impact of advice on non demonstrated states and on the correction of contradictory demonstrations. The proposed method will be implemented on a simple grid-world problem and a more complicated grasping problem, with a large discrete state and action spaces to test the algorithm effectiveness.

II. Related Work

A decision system is often implemented throughout the literature using Reinforcement Learning (RL) concepts. In RL, a policy is obtained using a reward function, which works as the consequence of performing a certain action a from a state s. However, in Learning from Demonstration (LfD) the goal is the opposite: obtain a reward function using a set of demonstrations given by the user, and then update the policy with the obtained reward function. This concept is particularly useful when the reward function is unknown or hard to define.

A. IRL in large state spaces

IRL is the most used method to approach the problem of LfD. *Russell et al. (2000)* [4] introduces the basic concepts of IRL. They propose three algorithms for IRL: two deal with cases when policy is entirely known, although one of them considers a finite state space and the other considers a linear approximation of the reward function over a continuous state space; the third algorithm deals with the case when the policy is partially known only through a finite set of observed trajectories. In all three algorithms it is proposed a linear programming formulation for the IRL problem in order to pick a reward function that differentiates the observed policy from the other (possible) suboptimal policies.

However, performing IRL for large state spaces is a very difficult task, since problems with too many components and constraints will make the computation too heavy to be feasible. *Russell et. al (2000)* [4] also presents the first solution for performing IRL in large state spaces. In their framework, the reward function is presented as a linear combination of previously known state functions, and the goal is to determine the weights for each component in order to complete the entire reward function based on the Bellman Optimality Equation. Although this solution substantially reduces the number of features to be computed, the number of constraints is still an unsolved issue. Other works were also developed in order to reduce the impact of the increase in the state space in computational terms, with *Burdick & Li (2017)* [3] being the one with the most success in this reduction. To avoid solving the computationally expensive reinforcement learning problems in reward learning, this work computes a function approximation method to ensure that the Bellman Optimality Equation always holds, and then obtains a function to maximize the likelihood of the observed demonstrations (known as MLIRL). The time complexity of the proposed method is linearly proportional to the cardinality of the action set, thus it can handle large state spaces efficiently.

B. Active Learning applied in IRL

Active Learning (AcL) consists of choosing a set of the most informative unlabelled features to be labelled, in order to minimize the amount of data necessary to train a model. The strategy of giving queries to an oracle (i.e. the demonstrator) so he can label them is a practical and useful method in modern learning literature. There are situations in which unlabeled data is abundant but manual labeling is expensive. In such a scenario, learning algorithms can actively query the user for labels. With AcL it is possible to obtain the same results with less necessary labelled data. Those queries can be chosen using several different strategy frameworks based on informativeness analysis. *Lopes et al. (2009)* [5] introduces the idea of using AcL to update the policy recursively, asking the demonstrator about new samples that involve more uncertain states. When choosing samples that are more targeted to certain states, a policy is learned more quickly with less necessary data. However, the approach taken in [5] is still not fast, as a single demonstration will not make a short-term difference in the update of the policy, making the convergence towards an optimal policy still slow.

C. Advice Learning: learning by advice vs learning by demonstrations

Kunapuli et al. (2013) [6] proposed a new extension for IRL by introducing the concept of Advice Learning (AdL) in IRL problems. AdL consists of assigning preferences to certain features (which can be actions, states or even rewards). Two subsets of features are then created: a subset of preferable features **Pref** and a subset of features to avoid **Av**. These preferences can be applied in three different ways:

- *Action Preferences* assigns preferred actions and to be avoided by the robot for a certain state.
- *State Preferences* gives preference to a certain subset of states to be explored.
- *Reward preferences* express preferences for certain immediate rewards, rather than their long-term values.

The combination between AdL and IRL allows the IRL algorithm to be performed through not only demonstrations but also the set of advice. This approach was proposed to minimize two assumptions that exist in LfD problems: demonstrator with optimal decisions and a sufficiently large set of demonstrations. In fact this approach is proved to lead

to better results with noisy demonstrations [6]. However, the algorithm proposed has some practical problems, specially the computational cost of giving advice to all states that exist in the space state.

Odom and Natarajan (2016) [2] propose a strong solution to this issue. They applied the idea of choosing queries to obtain new specific features. This approach allows not only greater flexibility in the information given by the human annotator to the agent, but also allows a better interaction with him, since the advice method is less prone to human errors. This flexibility also allows the attribution of a greater amount of information to the agent for each interaction. In addition, the algorithm proposed in [2] also allows the division of the state space into clusters according to the similarity in state's characteristics. With this clustering in mind, the query that is chosen will be a cluster of states instead of a single state. These clusters allow us to capture more information in each interaction, since the expert's answer for all the cluster is very likely to be the same.

III. Background

In this section it will be given a brief feedback about the fundamental theory for this work. Several notations and learning components will be addressed and it will also be discussed the limitations of some algorithms with large state spaces. The algorithm described in *Burdick & Li (2017)* [3] will also be explained, as it is fundamental for handling IRL in large state spaces.

A. Notation

A finite MDP is a tuple (S, A, P, γ, R). Here, S is a discrete set of states, A is a discrete set of actions, P is the state transition model, where P_{sa} is a vector of the state transition probabilities upon taking action a in state s. $\gamma \in [0, 1]$ is the discount factor and R is a vector where in position s is represented the reward associated to state s. In IRL context, R is unknown and a finite set of demonstrations is given. The set of demonstrations D consists of a set of state-action pairs that is supposed to fulfill the target optimal policy. A policy is a mapping $\pi : S \rightarrow A$ that determines how the agent acts under the MDP. The values of a state s, and the values of a state-action pair (s, a) under π are given by the following Bellman equations:

$$V^{\pi}(s) = \sum_{s'} P_{sa^*}(s') \times (R(s') + \gamma \times V^{\pi}(s')) \quad , \quad (1)$$

$$Q^{\pi}(s, a) = \sum_{s'} P_{sa}(s') \times (R(s') + \gamma \times V^{\pi}(s')) \quad , \quad (2)$$

where a^* is the optimal action for a state s. The policy is then updated using the Bellman optimality, i.e., the action mapped in the policy π for a given state s is the one with the highest action value for that state. From Bellman's equations, we can guarantee that a given policy π is optimal iif the reward function R satisfies the following inequality [4]:

$$(P_{a^*} - P_a) \times (I - \gamma \times P_{a^*})^{-1} \times R \succeq 0 \quad . \quad (3)$$

P_a is a transition matrix from state s to state s', given the action a. The symbol $\succeq$ means that all inputs to the resulting vector must be non-negative.

B. Advice as complement to demonstrations

To minimize the effects of a poor demonstration set (both in quantity and quality), *Kunapuli et al. (2013)* [6] introduced the idea of using advice as a complement to demonstrations so that fewer demonstrations are needed and minimize the problem of their inevitable sub-optimability. The advice will be given in the form of action preferences, which consist of two subsets of actions (preferable or avoidable) for each state. Logically, an action can not belong to both subsets. Advice allow us to give more comprehensive information about each state, which allows to cross off certain actions and prioritize others in search of the best action for each situation. That is why it is possible to use advice as a complement to LfD. The IRL algorithm proposed by *Kunapuli et al. (2013)* [6] is given by the following optimization problem:

$$\begin{aligned} \max_{R, \xi_i, \zeta_j} \quad & -||R||_1 + \lambda_d \sum_{i=1}^{N} \xi_i + \lambda_a \sum_{j \in S_e} \zeta_j \quad , \\ \text{s.t.} \quad & (P^i_{a^*} - P^i_a)(I - \gamma P_{a^*})^{-1} R \geqslant \xi_i, \quad \forall a \in A \setminus a^*, \\ & (P^j_a - P^j_{a'})(I - \gamma P_{a^*})^{-1} R \geqslant \zeta_j \,, \quad \begin{matrix} \forall a \in Pref_j \\ \forall a' \in Avd_j \end{matrix} , \\ & \zeta_j \geqslant 0 \,, \quad \forall j \in S_e, \\ & R_{max} \geqslant |R_i| \,, \quad \forall i = 1, \ldots, N. \end{aligned} \quad (4)$$

P^i_a is the transition probability row vector of a state i taking the action a. P_a is the transition matrix from state s to state s', given the action a. a^* is the optimal action associated to state i. N is the total number of states, while S_e is a subset with all states that contain advice. R_{max} is the maximum value that a reward can reach, while R_i is the reward associated to state i. ξ_i and ζ_j are variables that are intended to maximize. The goal of these variables is to maximize the minimum value of the constraints, for the demonstrations and advice respectively. $Pref_j$ and Av_j are the sets of preferable and avoidable actions for the state j respectively. λ_d and λ_a are adjustable parameters that define the degree of confidence that one has in relation to the demonstrations and advice given respectively. Finally, the l_1-regularizer in R helps to learn sparse rewards.

C. IRL for infinite state spaces

Although the concept of advice helps in the presence of insufficient demonstrations (both in quality and quantity), we still have a troubling problem for large state spaces, as there will certainly be problems in linear programming due to excessive constraints and excessive dimension of the variables involved in the problem. Furthermore, it will be very expensive to optimize such a large reward function. One way

of trying to solve these limitations would be to consider the reward function as a linear combination of base state functions. Assuming that $\Phi = \phi_1, \ldots, \phi_d$ are fixed, known bounded basis functions mapping the continuous state space that evaluate a certain state according to some criteria, the new aim of IRL is to find a set of weights $\Omega = \omega_1, \ldots, \omega_d$ in order to fit the reward function to a certain policy, assuming that:

$$R(s) = \omega_1 \times \phi_1(s) + \ldots + \omega_d \times \phi_d(s) \quad . \tag{5}$$

By the linearity of expectations, the value function V^π will also be a linear combination such as:

$$V^\pi(s) = \sum_{k=1}^{d} \omega_k \times V_k^\pi(s) \quad , \tag{6}$$

where V_k^π is the value function when the reward function is $R = \phi_k$. However, this formulation maintains the problem of excess constraints in the optimization problem. The simplest solution will then be to reduce the number of constraints by considering only a subset of states. This is possible due to the fact that the reward function is now continuous, which makes it possible to compute it even for states not considered in the optimization problem. The subset of states S_0 must therefore be as representative as possible of all possible situations in the environment. The continuous state space IRL algorithm proposed by *Ng. & Russell (2000)* [4] is then given by:

$$\begin{aligned} \max_{\Omega} \quad & \sum_{s \in S_0} \min_{a \in A \setminus a^*} p\left(\sum_{s'} \left(P_{sa^*}(s') - P_{sa}(s')\right) V^\pi(s') \right) \\ \text{s.t.} \quad & |\omega_k| \leqslant 1\,, \qquad k = 1, \ldots, d \quad , \end{aligned} \tag{7}$$

where p is given by $p(x) = x$ if $x \geqslant 0$ and $p(x) = 2x$ otherwise, and penalizes violations of the constraints.

Although this adaptation allows a more accessible computation of the IRL problem, the computational cost continues to be excessively high for the problem at hand.

D. Maximum likelihood IRL via Function Approximation

Although the IRL methods mentioned in the previous sections present good results for small state spaces, the same does not happen for large state spaces. Algorithms based on linear programming always end up having too many constraints in order to solve the problem in a reasonable time horizon. In order to handle large state spaces, *Burdick & Li (2017)* [3] propose a function approximation method to avoid this issue by ensuring that the Bellman Optimality Equation always holds and by estimating a function to maximize the likelihood of the current demonstrations.

Given the action space and the transition model, a reward function leads to a unique optimal value function. To learn the reward function from the observed demonstrations, instead of directly learning the reward function, they use a parameterized function, named as VR function, to represent the summation of the reward function and the discounted optimal value function:

$$f(s, \Theta) = R(s) + \gamma \times V(s) \quad , \tag{8}$$

where Θ denote the parameters of the VR function. Substituting (8) into (1) and (2), we will obtain equations of V and Q that only depends on the VR function. If we apply the Maximum Likelihood IRL (MLIRL) algorithm using the VR function approximation, the likelihood of the demonstrations is given by [3]:

$$\begin{aligned} L(D|\Theta) = & \sum_{(s,a) \in D} \beta \sum_{s'} P_{sa}(s') f(s', \Theta) - \\ & - \log \sum_{a' \in A} e^{\beta \sum_{s'} P_{sa'}(s') f(s', \Theta)} \quad , \end{aligned} \tag{9}$$

where β is a Boltzmann constant. Using a gradient ascent method to update the parameter Θ until reaching convergence allow us to obtain a VR function that ensures the Bellman Optimality condition and that maximizes the likelihood of the demonstrations. To compute $f(s, \Theta)$ we will assume that the VR function is a neural network where Θ are the weights and biases of the neural network and $L(D|\Theta)$ is the Loss function.

IV. Methodology

Initially the MDP is presented, where the entire state space, possible actions for each state and a transition model are defined. The grid-world problem consists on a grid where the goal state is the center of the grid. Each cell is a state, and the possible actions correspond to right, up, left and down. This grid will have several different sizes. The grasping problem consists on tabletop scenario with a robotic hand with only 2 fingers that can grasp several objects. The goal of the problem is to successfully grasp a certain object, and then move it to a certain goal pose according to some criteria defined by the demonstrator. The robot used to perform the grasping problem was the *KINOVA Gen3*, a simple 7-DoF manipulator with an integrated gripper with 2 fingers (*robotiq_2f_85*). The table has 0.4x0.4x0.5 meters and the robot is placed on top of a 0.5 meters high wooden box, such as illustrated in Fig. 1.

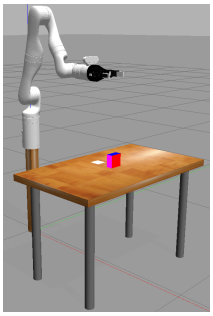

Fig. 1. Gazebo Environment with *KINOVA Gen3*.

A. MDP Definition

The MDP of the grid-world problem consists of a discrete finite set of states, which represent each cell of the grid, and

a set of actions that consists of moving to left, right, up or down.

The MDP of the grasping problem consists of a discrete finite set of states, which represent a description of the object to be grasped and its arrangement in the environment, and a set of actions that consists in a first instance of the possible grasping options, and then the options to rotate or move the robotic arm to a specific goal-state. The problem description was motivated by the work of *Coelho (2013)* [7], which proposes an MDP and a simple environment to be used in planning grasp and push tasks. The environment consists of a table, objects placed on top of the table and the manipulator used. This environment will be simulated using *Gazebo* simulation environment.

The constructed environment in this work also contains a Reachability Map, which consists of a table where for each cell it is indicated whether there is the capacity to perform grasp. Reachability Map is useful to limit the number of states and even actions of the problem, in addition to ensuring that the robotic arm does not enter unstable areas.

1) State Space: The state in the grid-world problem is the description of a single cell, which has features $s = [x, y]$, $x, y \in [0, 1]$.

The state in the grasping problem is the description of the object in the environment. Each state has the following parameters: pose and current situation of the grasp (whether or not the gripper is already grasping the object). If the object is being grasped by the gripper, the gripper orientation is added as state feature too. A 4x4x4 cm cube is always considered as the target object to be grasped. The object's reference frame has the same orientation as the world reference frame.

The first component of the state corresponds to the object's pose. The position of the object corresponds to the cell where the object origin is situated. Since this component must be also discretized, we will impose x as {0.5, 0.55, 0.6}m, y as {-0.1, 0.0, 0.1}m and z as {0.55, 0.6}m, giving a maximum of 18 positions. The orientation of the object will be discretized to obtain 48 different combinations. Table I identifies all possible orientations of an object in Z-Y-X Euler notation.

TABLE I
DIFFERENT ORIENTATIONS OF THE OBJECT, REPRESENTED BY THE ROTATION ANGLES α, β AND γ (IN DEGREES).

Rotation index (i)	α	β	γ	Reference axis
$i = 0, \ldots, 7$	0º	0º	45º i	z up
$i = 8, \ldots, 15$	90º	0º	45º (i - 8)	y up
$i = 16, \ldots, 23$	0º	270º	45º (i - 16)	x up
$i = 24, \ldots, 31$	180º	0º	45º (i - 24)	z down
$i = 32, \ldots, 39$	270º	0º	45º (i - 32)	y down
$i = 40, \ldots 47$	0º	90º	45º (i - 40)	x down

The second component of the state is a condition *is_grasp* that informs whether the robot is already grasping or not. This component is crucial to differentiate the possible set of actions that the state can take. If *is_grasped* is true, another component is added to the state: the gripper orientation, represented as a discretization of 18 possible orientations of the gripper (see Table II). In short, the state space is defined by a set of states:

- $Ori = \{0, \ldots, 47\}$: object orientation.
- $Pos = \{0, \ldots, 17\}$: object position on the table grid.
- $is\_grasp = \{False, True\}$: if the grasp is already made.
- $Grip = \{0, \ldots, 17,\ is\_grasp = True\}$: gripper orientation.

In conclusion, the number of possible states of the problem is **15984**.

2) Action Space: For the grid-world problem, an action corresponds on moving to other cell to the right, left, up or down.

For the grasping problem, the set of actions for each state will depend of the condition *is_grasp*. There are two main different sets of actions that are assigned depending of the second component of the state. If *is_grasp* is false, it means that the object is not already grasped. In that case, the set of actions consists of 18 possible grasps, as represented in Table II. Note that the angles in Table II are represented in X-Y-Z Euler notation. The gripper's reference frame is also the same as the world reference frame, for the sake of simplicity.

TABLE II
DIFFERENT GRIPPER ORIENTATIONS.

Rotation index (i)	roll	pitch	yaw
$i = 0$	$0°$	$90°$	$0°$
$i = 1, 2$	$-90°$	$45° \cdot (i - 1)$	$90°$
$i = 3, 4$	$90°$	$45° \cdot (i - 3)$	$-90°$
$i = 5$	$0°$	$135°$	$0°$
$i = 6$	$-135°$	$0°$	$90°$
$i = 7$	$-135°$	$45°$	$125°$
$i = 8$	$135°$	$0°$	$-90°$
$i = 9$	$135°$	$45°$	$-125°$
$i = 10, \ldots, 17$	$0°$	$180°$	$45° \cdot (i - 10)$

If *is_grasp* is true, then it means that the robot has already performed a grasp successfully, which means that now the goal is to move the grasped object to a goal state that will contain a particular pose that the object must end. Therefore, the action can be to change the rotation of the gripper, change the position of the gripper or drop the object. Translation and rotation action's features correspond to the gripper's goal position and orientation respectively. The number of rotation actions correspond to the number of possible gripper orientations. In total there are **55** possible actions in the defined MDP: 18 of grasping an object ($Grip = \{0, ..., 17\}, is\_grasp = False$), 18 of gripper translation ($Pos = \{0, ..., 17\}$), 18 of gripper rotation ($Grip = \{0, ..., 17\}, is\_grasp = True$) and 1 object drop.

3) State transition model: The state transition model corresponds to the probability that a given state s goes to s' taking action a. This model characterizes the environment and therefore it is also part of the MDP.

For the grid-world problem, the transition model gives a 100% chance that the transitioned state is expected (that is, if the action goes to the right, the next state will certainly be the cell to the right of the current one).

For the grasping problem, the transition model is initiated by Bernoulli experiments on the *Gazebo* simulator. Since the state space is large, it will be very difficult to perform

the Bernoulli experiments to all **15984** states, so it will be created a Kernel function so that many states can be trained to each experiment, based on state's similarity. This process is repeated, and normalized to a probability value. Both the grid-world and grasping transition models are stored in sparse matrices due to their large size.

B. Proposed Learning Algorithm

The proposed algorithm will now be introduced. The aim is to obtain a policy π that decides the best action for each situation in order to reach the goal state with maximum reward. A scheme of the proposed learning algorithm is shown in Fig. 2.

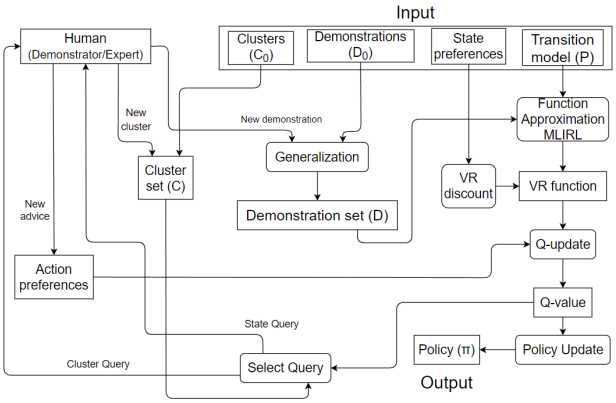

Fig. 2. A framework for the algorithm proposed.

Since we are working with a very large state space, a suitable solution to the IRL problem will be to consider a VR function as a function approximation of the summation between the reward function and the discounted state value function such as (8). As the VR function is an approximate function, it can be computed from the optimization of the parameters of a neural network using a gradient ascent method. To perform this method we will use a neural network to optimize a parameter set Θ in order to get the VR function that maximizes the log likelihood of the demonstrations, as performed in *Burdick & Li (2017)* [3]. However, a set of AcL and AdL approaches will be added to this method, asking the demonstrator for new "optimal" responses and advice to less certain states. A predefined set of state preferences is given in both problems. For the grid-world problem, it will be preferred to go to the center and the limit states will be avoided. Subsubsection IV-C3 explains the state preference framework for the grasping problem. Once the final VR function is found, a 20% benefit/discount is given to the VR values of the preferable or avoidable states respectively. A demonstration set D and a set of $clusters$ must also be given to the algorithm. In this framework, a demonstration is represented as a sequence of state-action pairs and should be considered as optimal (explained in more detail in Subsubsection IV-C1). A set of $clusters$ would be given in the beginning, as in this case it is quite easy to know some states with similar behavior in policy (explained in more detail in Subsubsection IV-C2). Among the large state space, two queries are then chosen to be labelled: a state query for demonstration labelling and a cluster query for advice labelling. The less certain state will be labelled by the demonstrator, and later added to the demonstration set after a generalization process based on a Kernel function (explained in Subsubsection IV-C1). This choice is made through measures of informativeness. Uncertainty sampling is the most widely used measure in the literature on a large scale and also the one that will be used on this approach. The uncertainty technique itself involves different strategies, but one of the most used is undoubtedly entropy, which is given by the following expression:

$$E(s) = -\sum_{a \in A} P_s(a) \cdot \log P_s(a) \quad . \tag{10}$$

In this case, $E(s)$ is the uncertainty w.r.t. policy π, so the probability $P_s(a)$ will be given by the following expression:

$$P_s(a) = \frac{e^{Q(s,a)}}{\sum_{b \in A} e^{Q(s,b)}} \quad . \tag{11}$$

In addition to the most uncertain states, the most uncertain cluster will also be queried in order to complement the information provided with action advice given by an expert. As in *Odom & Natarajan (2016)* [2], an expert will be responsible for providing advice in form of action preferences for a given cluster. These actions will be held in preferred or avoidable action sets, and a 20% benefit or 100% discount (i.e. the action is excluded of the equation) is given to the action value $Q(s, a)$, when updating the matrix Q. No action can belong to both subsets. Algorithm 1 explains in pseudo-code the proposed framework.

This framework manages to combine the idea of obtaining a policy in a more accelerated way from a method of function approximation with the idea of providing more information in quality and quantity in an iterative way, thus accelerating the learning process. Furthermore, the state and action preferences do not affect the training of the neural network, as the VR function only depends on the demonstrations given for the Loss function. The given advice serves only as a push for the less certain states after obtaining the VR function. In this way, both processes are carried out independently, so that there is no interference between them.

C. Particular definitions for the grasping problem

In this section, descriptions of certain data acquisition processes for the grasping problem will be covered, such as the process of obtaining the initial set of demonstrations and clusters and the definition of state preferences.

1) Demonstration set: As previously mentioned, the demonstration set in this context consists of a set of state-action pairs with considered optimal behavior. Each state-action pair is called a trajectory. For this work, the construction of the trajectories for the first demonstration set performs the following steps:

Algorithm .1: Active FA-MLIRL algorithm

Function *FA-AL-MLIRL(Budget, Demonstrator, Expert, clusters, state_pref, D_0, P):*

- $D = D_0$;
- $advice = \emptyset$;
- create variable Θ_0 as neural network parameter;
- build $f(\Theta)$ as the output of the neural network;
- initialize Θ_0 arbitrarily;
- Q, Θ = FunctionApprox(D, P, Θ_0);
- **for** $k = 1$ *to Budget* **do**
  - $state\_query$, $cluster\_query$ = SelectQuery(Q, $clusters$);
  - New_D = Demonstrator($state\_query$);
  - New_D = Generalization(New_D);
  - D.append(New_D);
  - new_advice = Expert($cluster\_query$);
  - $advice$.append(new_advice);
  - Q, Θ = FunctionApprox(D, P, Θ);
- **end**
- π = PolicyUpdate(Q);
- return π;

end

Function *FunctionApprox(D, P, Θ, $state\_pref$, $advice$):*

- Compute neural network using Equation (9) as loss function $L(D|\Theta)$;
- **while** Θ *not converging* **do**
  - Θ = Θ + α $\nabla_\Theta L(D|\Theta)$;
- **end**
- $f(\Theta)$ = DiscountState($f(\Theta)$, $state\_pref$);
- Q = UpdateQ($f(\Theta)$, $advice$);
- return Q, Θ;

end

Function *SelectQuery(Q, $clusters$):*

- **for** $s \in S$ **do**
  - E(s) = 0;
  - **for** $a \in A$ **do**
    - $P_s(a)$ = $\frac{e^{Q(s,a)}}{\sum_{b \in A} e^{Q(s,b)}}$;
    - $E(s)$ -= $P_s(a) log P_s(a)$;
  - **end**
- **end**
- **for** $c \in clusters$ **do**
  - $U(c) = \frac{\sum_{s \in c} E(s)}{len(c)}$;
- **end**
- $state\_query = argmax_s\ E$;
- $cluster\_query = argmax_c\ U$;
- return $state\_query$, $cluster\_query$;

end

Function *DiscountState($f(\Theta)$, $state\_pref$):*

- **for** $s \in S$ **do**
  - **if** $s \in state\_pref.pref$ **then**
    - $f(\Theta, s) = f(\Theta, s) + 0.2 \times |f(\Theta, s)|$;
  - **end**
  - **if** $s \in state\_pref.avd$ **then**
    - $f(\Theta, s) = f(\Theta, s) - 0.2 \times |f(\Theta, s)|$;
  - **end**
- **end**
- return $f(\Theta)$;

end

Function *UpdateQ($f(\Theta), advice$):*

- **for** $s \in S$ **do**
  - **for** $a \in A$ **do**
    - $Q(s,a) = \sum_{s'} P(s,a,s') \times f(\Theta, s')$;
    - **if** $a \in advice.pref(s)$ **then**
      - $Q(s,a) = Q(s,a) + 0.2 \times |Q(s,a)|$;
    - **end**
    - **if** $a \in advice.avd(s)$ **then**
      - $Q(s,a) = -2$;
    - **end**
  - **end**
- **end**
- return Q;

end

Function *PolicyUpdate(Q):*

- **for** $s \in S$ **do**
  - $\pi(s) = argmax_a\ Q(s)$;
- **end**
- return π;

end

- The object is placed somewhere on the table. An attempt is made to grasp that object. The choice of which action to take is made by the demonstrator which will choose the considered optimal grasp action for that situation;
- If the object was successfully grasped, the next action to be taken will be rotation, translation or dropping the object. This sequence of events is done until the object is no longer grasped;
- If the object was not grasped, a new grasp action will be experimented;
- Repeat the previous steps until the object is in the goal state (non grasped object with pose [0.5m, 0m, 0.55m, 0°, 0°, 0°]). After reaching the goal state, a new object will be randomly placed and all the process is repeated successively.

Considering the large size of the state space, it is unaffordable to perform an initial set of demonstrations large enough to cover a large part of the state space. Then, a generalization

of the trajectories will be performed. When querying the more uncertain states to the demonstrator, it responds with the action it thinks is optimal for that state and the resulting demonstration is again submitted to a generalization. Thus, let us consider the generalization criteria:

- For states where the object is not being grasped, the same grasping action applies wherever the object is, as long as the object's orientation remains, except when the gripper points left at $y = -0.1$m or points right at $y = 0.1$m (since the gripper is less comfortable in those poses).
- If on a given trajectory the action is to translate the object to $x = 0.5$m and $y = 0.0$m (target position) or to drop it, the same action is expected for any state that maintains all state components of the trajectory except the x and y positions, as long as it does not present one of the following less comfortable poses of the gripper:
  - The gripper points left at $y = -0.1$m;
  - The gripper points right at $y = 0.1$m.
- The same happens for any rotation, as long as both the initial state and the predicted final state meet the requirements presented here.

2) State clustering: In order for the expert to be able to advise certain actions for a set of states instead of an individual state, it is necessary to perform a pre-grouping of all the problem states in clusters according to their similarity. In this way, learning the decision model will be performed much more quickly in relation to advice, and will maintain performance if this grouping is perfect. It is very unlikely that the grouping of states will be perfect (i.e., the consequence would be exactly the same if a certain action is taken for two similar states), so it is necessary to find a balance. Taking this into account, several clusters were created in order to amount the various states according to a certain feature. Table III shows the description of all 20 primordial clusters of the problem.

TABLE III
DESCRIPTION OF EACH CLUSTER.

Feature	Description
Division by *is_grasp*	0
	1
Division by gripper orientation	pointing down: $R[2][2] < -0.9$
	pointing front: $R[0][2] > 0.9$
	pointing right: $R[1][2] < -0.9$
	pointing left: $R[1][2] > 0.9$
Division by object orientation	$x = 0°$ or $x = 180°$, $y = 0°$
	$x = 0°$, $y = 90°$ or $y = 270°$
	$x = 90°$ or $x = 270°$, $y = 0°$
	$z = 0°$ or $z = 180°$
	$z = 45°$ or $z = 225°$
	$z = 90°$ or $z = 270°$
	$z = 135°$ or $z = 315°$
Division by table zones	back limit: $x = 0.5$m
	front limit: $x = 0.6$m
	right limit: $y = -0.1$m
	left limit: $y = 0.1$m
	under limit: $z = 0.55$m
	upper limit: $z = 0.6$m
	center: $x = 0.55$m and $y = 0$m

3) State preferences: For the problem at hand, there are states where it is easy to see if they are good or bad. Conversely, certain gripper poses are not good in certain areas of the table, so this outcome should be avoided. The choice of preferred and avoidable states is a form of information that can be used as a complement to demonstrations and action preferences. This form of advice will be known at the beginning of the algorithm and will not undergo any querying process. A certain state will belong to the preferred group if it meets the following requirements:

- Object orientation is $(0°, 0°, 0°)$;
- Object position is (0.5m, 0.0m, z).

Likewise, a state will belong to the group to be avoided if it meets the following requirements:

- There is no way to transition the state to another where the object's orientation is $(0°, 0°, 0°)$;
- Object at $y = 0.1$m and gripper pointing to the right;
- Object at $y = -0.1$m and gripper pointing to the left;

V. RESULTS

This section is addressed to show and explain the experiments performed in this work.

A. Grid-world problem

Fig. 3 denotes the grid-world problem with the given demonstration set. The green cell is the goal state, the red arrows represent the demonstration set, the yellow cells are the non-demonstrated states. In this setting, the states to avoid are the limits of the grid and the preferred state is only the goal state. We can observe that this new information can have a lot of weight in the final policy learned, as it can even surpass the information given in the demonstrations. In Fig. 4 it is possible to observe that, with state preference advice, the trajectories move away from the limits of the grid, even when the demonstrations say so. This example shows that state preferences can fix suboptimal demonstrations.

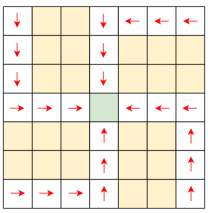

Fig. 3. Demonstrations given to 7x7 grid. Red arrows are the given demonstrations. The yellow cells are the non-demonstrated states.

Then, experiments were performed to evaluate the scalability of the algorithm as the number of states increased. The same experiments were done on 25x25, 35x35 and 101x101 grids. The neural network used for the 3 different state spaces has the same format: 2 hidden layers with *ReLU* activation function, an output layer with *Tanh* activation function and learning rate $\alpha = 0.01$. Each neural network input consists of a vector of 2 values that correspond to the 2 features of this problem: $s = [x, y]$, $x, y \in [0, 1]$. For these experiments,

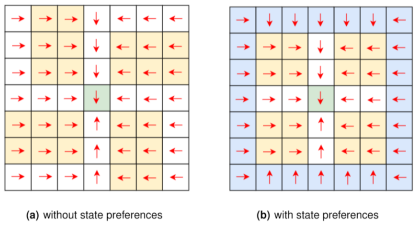

Fig. 4. Final policy of 7x7 grid. Red arrows are the given demonstrations. The yellow cells are the non-demonstrated states, the green cell is the preferred state and the blue cells are the states to be avoided in the state preference framework.

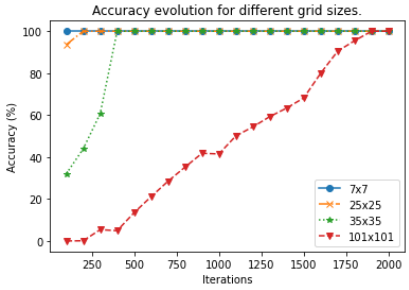

Fig. 6. Accuracy evolution for different grid sizes.

no advice module was used. Following the example of Fig. 3, demonstrations were given in order to perform the same 4 trajectories from a corner to the center of the grid. Fig. 5 shows the obtained policy for the 101x101 grid.

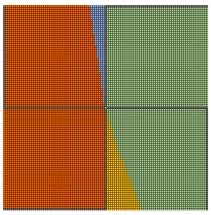

Fig. 5. Final policy of 101×101 grid. Orange: Right; Blue: Down; Green: Left; Yellow: Up; Gray: Demonstrations.

Throughout the iterations, runs of the grid-world environment will be performed with all cells as the initial state and following the policy obtained, and an accuracy score will be obtained for the number of runs that reached the goal state after N state transitions (where $N = 7$ for the 7x7 grid for example). Fig. 6 represents the evolution along the iterations of the policy's accuracy score for different grid sizes.

In Fig. 6, it is possible to conclude that, even with a decrease in the percentage of states demonstrated with the increase of the grid size, the algorithm always manages to reach excellent performances with the increase in the number of iterations in the proposed grid-world problem. For the 101x101 grid, only 3.92% of the demonstrated states were needed for the algorithm to reach an optimal policy in 1900 iterations. The choice of gray states in Fig. 5 also helped this result, since if only states to the right of the goal state had been demonstrated, the algorithm would hardly learn the states to the left of the goal state, for example. That is why AcL is also important in this framework: the fact that demonstrations were given scattered all over the grid allowed the algorithm to need fewer demonstrations to achieve 100% accuracy.

B. Grasping problem

In this section, the proposed algorithm is applied to the grasping problem. The network was trained from a set of 981 initial demonstrations (Subsubsection IV-C1) and after 10000 iterations an initial policy is obtained. Then, new demonstrations and new advice in action preference format will be introduced every 100 iterations to the most uncertain states/clusters labeled by the human demonstrator. All the advice in state preference format is given in the beginning. The neural network consists of two hidden layers with *ReLU* activation function, and the output layer has *Tanh* activation function, as performed in the grid case. Hidden layers have 150 and 50 neurons respectively. The input is given by 15984 states with 10 features each state, so giving 159840 entries from the network. The 10 features correspond to the state components normalized to values between 0 and 1: the first 6 features correspond to the pose of the object, the seventh feature corresponds to the *is_grasp* component (0 or 1) and the last 3 features correspond to the orientation of the gripper ([0,0,0] if the object is not grasped). It was noticed an increase in the computation time of each iteration compared to all grid-world cases, since the number of actions is greater. Three experiments were performed with different additional demonstration set sizes. In the first experiment, no demonstrations were added, so the network was trained only with the 981 initial demonstrations and with 10000 iterations. In the second experiment, additional demonstrations and advice in action preference format were given iteratively to random states and clusters. In the third experiment the additional demonstrations and advice in action preference format were given iteratively using the proposed query selection process based on uncertainty. For this problem, a validation set consisting of 100 demonstrations is built by the demonstrator. The accuracy score in this case will be the percentage of times where the states shown in this validation set match the policy in the optimal action. Table IV shows the obtained results for these three experiments.

TABLE IV
ACCURACY SCORE WITH OR WITHOUT STATE AND ACTION PREFERENCES FOR 3 DIFFERENT DEMONSTRATION SETS. THE FIRST LINE CORRESPONDS TO THE PERCENTAGE OF STATES DEMONSTRATED IN THE DEMONSTRATION SET.

	Init set	Set without AcL	Set with AcL
States demonstrated (%)	6.14	14.25	8.11
Without AdL (%)	8	15	19
Action AdL (%)	9	15	19
State AdL (%)	14	18	21
Action and State AdL (%)	15	18	21

Through Table IV, it is possible to observe the effect of choosing the query through AcL: even with fewer demon-

strations, the results of the third experiment network are better than those of the second experiment. This is because the demonstrations added in the third experiment are more informative as the states shown are more uncertain. In other words, the new demonstrations show behaviors that are more pronounced and less known by the neural network.

The inclusion of state and action preferences in the iterative process to update the Q matrix also improves the accuracy, as it consists in the introduction of information that is not covered by the demonstration set. Even so, the results are not as expected when compared to the grid case. These results may be caused by some error in the construction of the MDP that was not identified by the author, however there are other reasons identified:

- The transition model is not the most correct as it was not possible to perform enough Bernoulli experiments to obtain a robust and complete model.
- The fact that there are actions that can not be performed for certain states (such as grasping an object that is already being grasped) may also have an influence on the following states.
- Demonstrations may contain certain contradictions, as they are susceptible to labeling errors by the demonstrator or in the generalization process. These contradictions can lead to a more difficult interpretation of the intended behaviors in the final policy. However, the given advice was able to correct some of these possible errors by the demonstrator, as can be seen in the improved accuracy.
- The choice of the given states for demonstration may also not have been the best, since it is more difficult to identify desirable behaviors with the set of state features presented in the grasp problem. Also for this reason, improvements are expected with the use of a query choice technique that better identifies the states that need to be demonstrated for a better identification of these behaviors.

C. Discussion

In conclusion, although accuracy values are lower than expected for the grasping problem, it is possible to confirm certain aspects referred in both problems:

- The choice of the states that will be demonstrated is very important in obtaining the optimal policy. Choosing the right states allows capturing the most crucial information for the neural network to be able to identify the desirable behaviors.
- The completion of learning through demonstrations with advice allows not only to correct some suboptimal demonstrations, but also to obtain optimal actions for some states not included in the demonstrations, accelerating the process of obtaining a good policy.
- The neural network presents a higher computational cost with the increase in the number of actions, as seen in [3].

VI. Conclusion

The algorithm proposed in the work implement the idea on learning by demonstrations and advice simultaneously, in order to obtain more information from an expert with less iterations. The algorithm consisted on applying AcL and AdL on a MLIRL framework, obtaining an approximation of the VR function using a neural network and consequently a new updated policy. Then a query selection model based on uncertainty was implemented to obtain a cluster query that was given to the expert to label it in the form of action advice and a state query to label it in demonstration form.

Through the obtained results it was possible to observe the importance of the informativeness of the demonstrations in the proposed algorithm, since the choice of the right states to demonstrate will present better results with less necessary demonstrations (as is the case of grids). It was also possible to observe an increase in accuracy when using advice as a complement to the demonstrations.

A. Future work

After experimenting with the proposed algorithm to accelerate the learning process in two different scenarios, many other approaches can be improved or added to this work for the grasping problem case. The transition model can be improved by performing Bernoulli experiments for all the problem states a sufficient number of times to obtain a good representation of the behavior of the environment. In this work, it was not possible to compute the sufficient number of experiments through lack of time. Considering the difficulty in creating a good transition model, the ideal would be to successfully adapt the proposed algorithm to a model-free version by replacing the Q-matrix update formula in order to avoid the presence of a transition model P. Other possible improvements to this framework would be to improve clustering quality or implement a query selection strategy more appropriate for this context than uncertainty sampling. Finally, the baseline of the grasping problem could be transferred to the real world, thus implementing a visual component to identify the state (namely where the object is and if it is being effectively grasped) using the camera integrated in the wrist of the *KINOVA Gen3* manipulator. This algorithm can also be tested for an even larger state space, as there was no time to run the algorithm with a broader problem.

References

[1] B. Settles, "Active Learning literature survey," Computer Sciences Technical Report 1648, University of Wisconsin, Madison, 2009.

[2] P. Odom and S. Natarajan,"Active advice seeking for inverse reinforcement learning," in *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*, 2016, pp.512–520.

[3] K. Li and J. W. Burdick, "Inverse reinforcement learning in large state spaces via function approximation," 2017.

[4] A. Y. Ng, S. J. Russell et al., "Algorithms for inverse reinforcement learning." in *Icml*, vol. 1, 2000, p. 2.

[5] M. Lopes, F. Melo, and L. Montesano, "Active learning for reward estimation in inverse reinforcement learning," in *European Conference on Machine Learning (ECML/PKDD)*, 2009, pp. 31–46.

[6] G. Kunapuli, P. Odom, J. W. Shavlik, and S. Natarajan, "Guiding autonomous agents to better behaviors through human advice," in *2013 IEEE 13th international conference on data mining.IEEE*, 2013, pp. 409–418.

[7] R. Coelho, "Planning push and grasp actions: Experiments on the icub robot," 2013.